POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

PROJEKT Z BAZ DANYCH

Firma Transportowa

Link do kodu aplikacji: https://github.com/kapalzi/FirmaTransportowa

Termin zajęć: Poniedziałek, 9:15–11:00

AUTOR/AUTORZY:
Krzysztof Kapała
Indeks:235060
Wojciech Wątroba
Indeks:235554

PROWADZĄCY ZAJĘCIA:
dr inż. Roman Ptak, W4/K9

Wrocław, 2018 r.

1.Wstęp

Aplikacja będzie przeznaczona dla firmy transportowej działającej na terenie Polski, zatrudniającej nie więcej niż 5 kierowców, 2 pracowników, których zadaniem jest przydzielanie kierowców do realizacji zleceń i 1 zarządcę. Firma zajmuje się dostarczaniem paczek bezpośrednio do klientów. Klient składa zamówienie, następnie ma możliwość wyboru dogodnego dla siebie terminu dostarczenia nie wcześniej niż 1 dzień od daty złożenia zamówienia. Pracownik przydziela odpowiedniego kierowcę(ze względu na posiadane przez niego uprawnienia oraz dostępne pojazdy) i po potwierdzeniu wpłaty zlecenie jest realizowane.

Aplikacja będzie zawierać informacje o usługach swiadczonych przez firmę transportową. Będzie posiadać również informację o klientach i zamówieniach przez nich realizowanych, o dostawcach pracujących w firmie i dostępnych do wykorzystania pojazdach dostawczych, a także o opiniach wystawianych przez klientów po skorzystaniu ze swiadczeń oferowanych przez firmę. Zarządca bazy będzie miał dodatkowo wgląd w szczegółowe informacje o pojazdach i wynagrodzenia pracowników, które może dowolnie zmieniać. Dzięki temu programowi bez problemu można w pełni obsługiwać firmę transportową, kontrolować informacje nt. jej pracowników oraz klientów.

2.Wymagania Funkcjonalne

Klient

-możliwosć złożenia zamówienia z wyspecyfikowanym terminem dostarczenia odpowiednim dla klienta oraz miejscem dostarczenia wybranych dóbr.

-możliwość dokonania opłaty za usługę.

-możliwosć dokonania dodatkowej opłaty w celu przyspieszenia realizacji zamówienia.

-możliwość anulowania zamówienia.

Po złożeniu zamówienia pracownik przydziający kierowców ma możliwosć:

-wglądu w złożone zamówienia(oraz kierowców do nich przydzielonych) i terminy do których mają być dostarczone.

-wyboru pojazdu do transportu, w zależnosci od kategorii wagowej/gabarytowej przesyłki

-wyboru kierowcy do danego zamówienia ze względu na posiadane przez niego uprawnienia lub jego dostępnosć (czy nie wykonuje innego zlecenia)

-wysłania zamówienia do realizacji po uprzednim sprawdzeniu czy zostało opłacone

-możliwość anulowania zamówienia

Zarządca:

-możliwosć wglądu w listę pracowników, ich dane osobowe, pensje oraz datę zatrudnienia i pełnione stanowisko.

-możliwość zatrudniania nowych pracowników

-możliwość zwalniania pracowników

3.Wymagania niefunkcjonalne

3.1.1 Wykorzystywane technologie i narzędzia:

-Baza danych będzie wykorzystywała technologie MySQL oraz MySQL Workbench -Do stworzenia aplikacji wykorzystany zostanie C#

-Układ graficzny będzie uproszczony i łatwy do obsługi dla użytkowników

3.1.2 Wymagania dotyczące rozmiaru bazy danych:

-Do bazy będzie dodawane do 100 zleceń dziennie.

-Baza będzie przechowywała informacje o zatrudnionych w firmie pracownikach.

-Baza będzie przechowywała informacje o 6 pojazdach dostępnych w firmie.

3.1.3 Wymagania dotyczące bezpieczeństwa systemu:

-Tylko uprawnione osoby mają dostęp do list pracowników -

Każdy użytkownik będzie posiadał swoje hasło

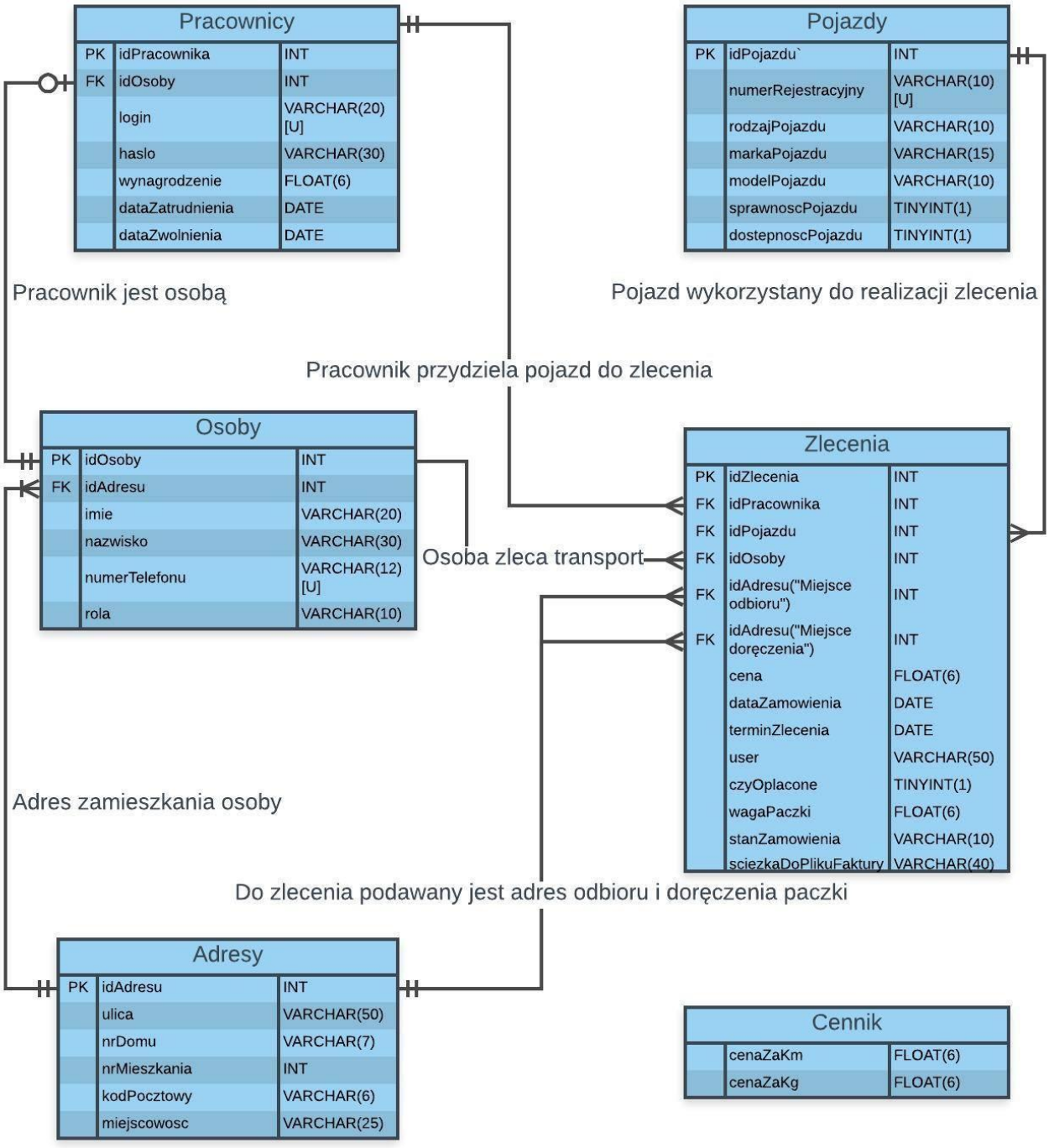
-Aplikacja będzie działać wyłącznie offline w celu zwiększenia bezpieczeństwa.

-Aplikacja będzie zainstalowana tylko na kilku komputerach w biurze firmy.

-Baza będzie przechowywana w dodatkowym pliku backupowym tworzonym codziennie aby w razie awarii nie utracić danych

4.1. Projekt bazy danych

4.1.1. Diagram ERD (model fizyczny, logiczny i konceptualny)



4.1.2. Ograniczenia integralności danych

```
Ograniczenia w tabeli Pracownik:
idPracownika: PRIMARY KEY
idOsoby: REFERENCES Osoba(idOsoby)
```

```
login : NOT NULL
haslo: NOT NULL AND CHECK(LENGTH(haslo) > 8)
wynagrodzenie: NOT NULL
dataZatrudnienia: NOT NULL
dataZwolnienia: DEFAULT ‘’

Ograniczenia w tabeli Osoba:
idOsoby: PRIMARY KEY
idAdresu: REFERENCES ADRESY(idAdresu)
imie: NOT NULL
nazwisko: NOT NULL
numerTelefonu: NOT NULL
rola: NOT NULL

Ograniczenia w tableli Adresy:
idAdresu: PRIMARY KEY
ulica: NOT NULL
nrDomu: NOT NULL
nrMieszkania: DEFAULT ‘’
miejscowosc: NOT NULL

Ograniczenia w tableli Pojazdy:
idPojazdu: PRIMARY KEY
numerRejestracyjny: NOT NULL
rodzajPojazdu: NOT NULL
markaPojazdu: NOT NULL
sprawnoscPojazdu: NOT NULL AND DEFAULT ‘1’
dostepnoscPojazdu: NOT NULL

Ograniczenia w tabeli Zlecenia:
idZlecenia: PRIMARY KEY
idPracownika: REFERENCES PRACOWNIK(idPracownika)
idPojazdu: REFERENCES POJAZDY(idPojazdu)
idOsoby: REFERENCES OSOBA(idOsoby)
idAdresu(„Miejsce odbioru”): REFERENCES ADRESY(idAdresu)
idAdresu(„Miejsce doręczenia”): REFERENCES ADRESY(idAdresu)
idCennika: REFERENCES CENNIK(idCennika)
dataZamowienia: NOT NULL AND DEFAULT ‘SYSDATE()’
terminZlecenia: NOT NULL
czyOplacone:NOT NULL AND DEFAULT ‘0’
wagaPaczki: NOT NULL
stanZamowienia: NOT NULL AND DEFAULT ‘W realizacji’
sciezkaDoPlikuFaktury: NOT NULL

Ograniczenia w tabeli Cennik:
idCennika: PRIMARY KEY
cenaZaKm: NOT NULL
cenazaKg: NOT NULL
wartoscKoncowa: NOT NULL
```

4.1.2. Ograniczenia integralności danych

```
WIDOKI:
Lista Pracowników – Imię,Nazwisko z tabeli Osoby, Wynagrodzenie z tabeli
Pracownicy

Lista Zamówień – Imię,Nazwisko z tabeli Osoby, ulica, nr domu, nr
mieszkania, miejscowość z tabeli Adresy - Adres Odbioru, ulica, nr domu, nr
mieszkania, miejscowość z tabeli Adresy - Adres Dostarczenia, Termin
Dostarczenia z tabeli Zlecenia
```

Triggery:

Trigger_doZaplaty – po złożeniu zamówienia ustawia stan zamówienia na „nieopłacone”
Trigger_wRealizacji – po opłaceniu zlecenia ustawia stan zamówienia na „w realizacji”

Indeksy:

Indeks jest strukturą danych, która znacząco przyśpiesza proces przeszukiwania tabeli. W tym celu należało przemyśleć, które pole są najczęściej brane pod uwagę podczas przeszukiwania.

Index name(imie,nazwisko) w tabeli Osoby – szansa na to że dwa rekordy w bazie będą zawierać to samo imię i nazwisko jest niewielka, więc taki indeks przyspieszy wyszukiwanie konkretnej osoby.

Index(dataZamowienia) w tabeli Zlecenia – umożliwi szybsze wyszukiwanie zamówień złożonych konkretnego dnia

Index(numerTelefonu) w tabeli Osoby
Index(numerRejestracyjny) w tabeli Pojazdy
Index(login) w tabeli Pracownicy

4.1.4. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

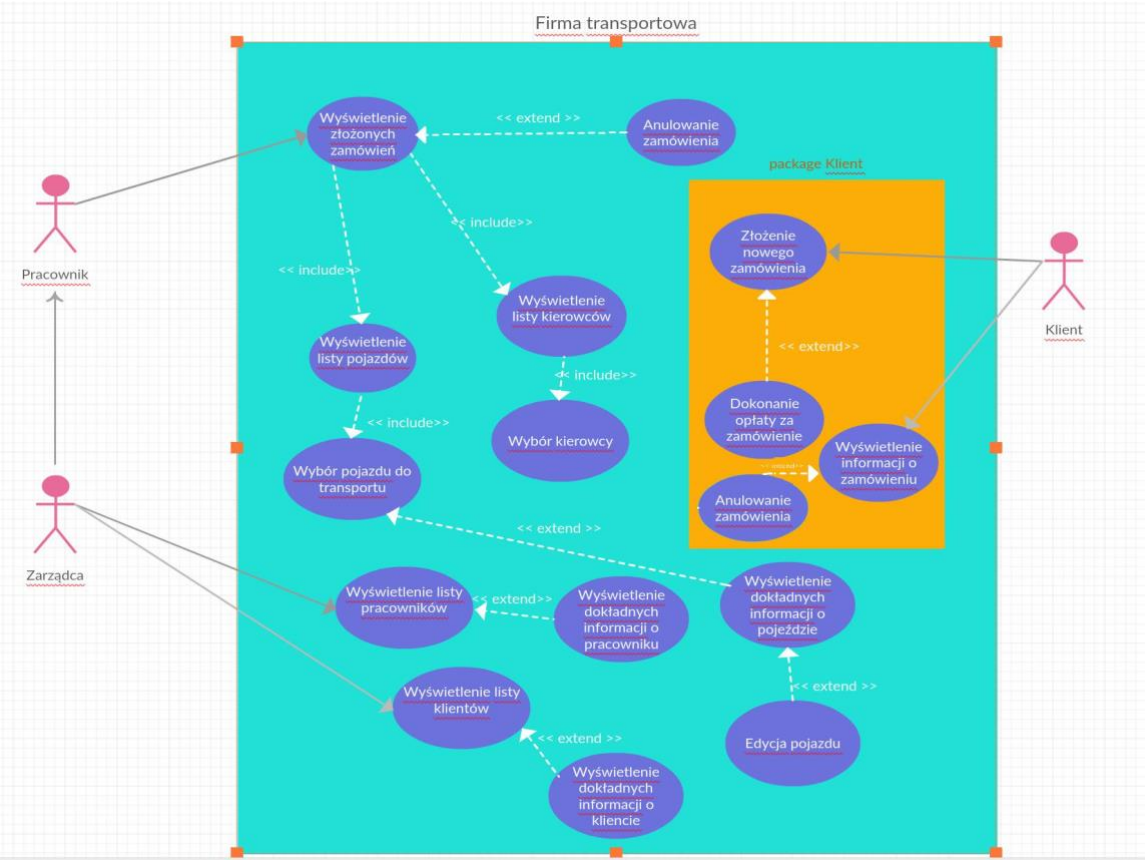
Baza będzie wymagała logowania w celu zwiększenia bezpieczeństwa.

Dostępność do tabel:

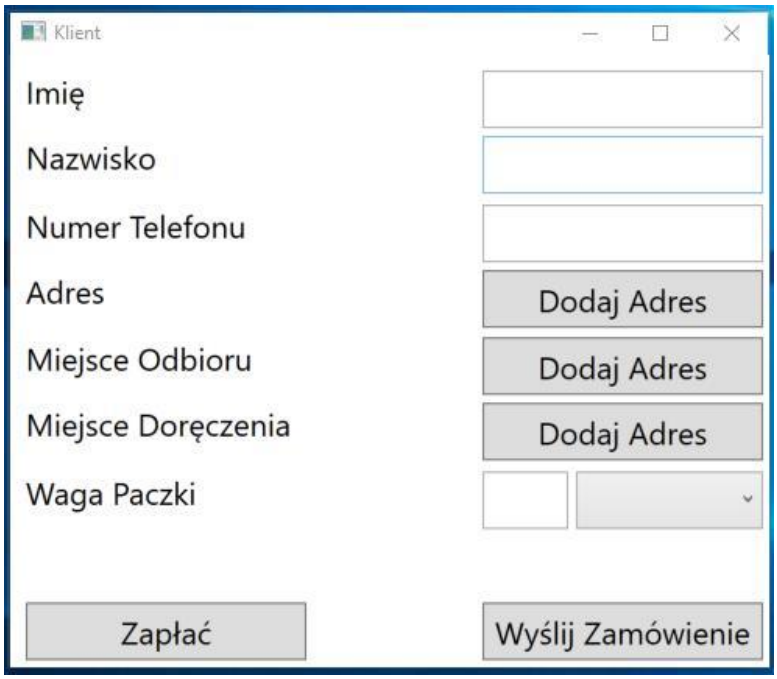
Pracownicy	Nie ma dostępu do tabeli.	Nie ma dostępu do tabeli.	Może dodawać, wyświetlać, modyfikować i usuwać.
Pojazdy	Nie ma dostępu do tabeli.	Może dodawać, wyświetlać, modyfikować i usuwać.	Może dodawać, wyświetlać, modyfikować i usuwać.
Zlecenia	Klient dodaje nowe zlecenie, może wyświetlić tylko te, które go dotyczą oraz ma możliwość anulowania go.	Może dodawać, wyświetlać, modyfikować i usuwać.	Może dodawać, wyświetlać, modyfikować i usuwać.
Adresy	Może wprowadzić swój adres lub zmienić istniejący.	Może wyświetlać wszystkie adresy	Może dodawać, wyświetlać, modyfikować i usuwać.
Cennik	Może wyświetlić aktualne stawki cenowe.	Może wyświetlić oraz zmodyfikować aktualne stawki cenowe.	Może dodać, usunąć, wyświetlić oraz zmodyfikować aktualne stawki cenowe.

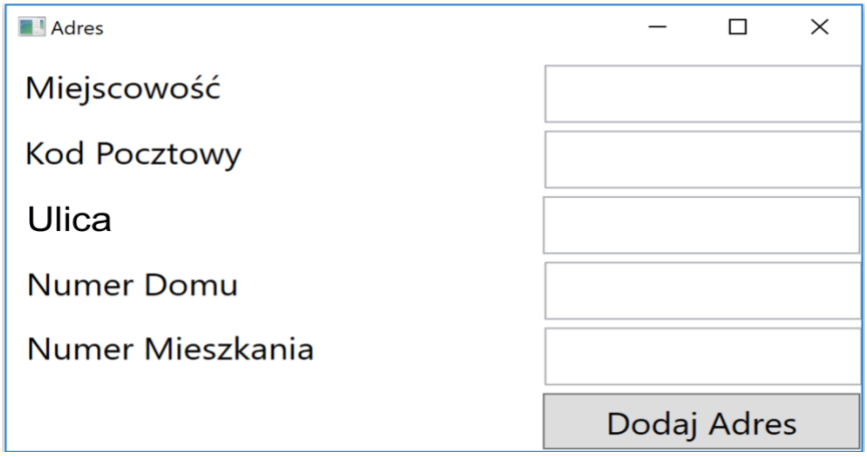
4.2. Projekt aplikacji użytkownika

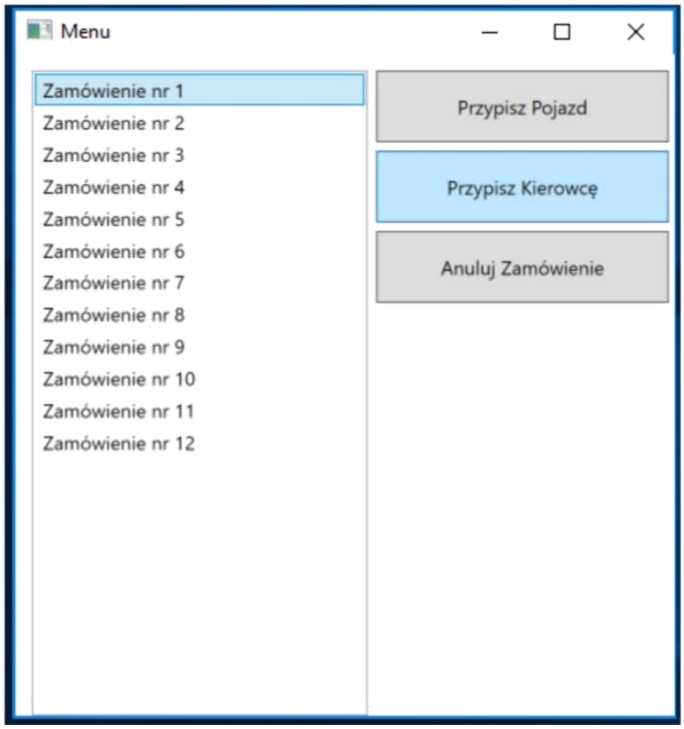
4.2.1. Architektura aplikacji i diagramy projektowe



4.2.2. Interfejs graficzny, struktura menu i wybrane funkcje.



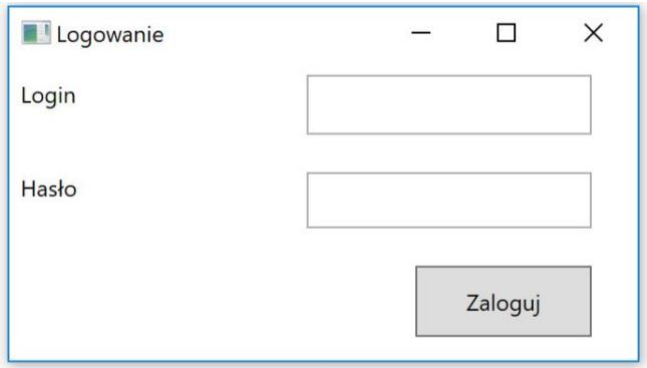




4.2.3. Metoda podłączania do bazy danych

– połączenie z bazą danych za pomocą ODBC (Open DataBase Connectivity)

4.2.4. Projekt zabezpieczeń na poziomie aplikacji



5. Implementacja systemu baz danych
5.1. Tworzenie tabel i definiowanie ograniczeń

Przy tworzeniu tabel korzystaliśmy z domyślnego silnika InnoDB.

```
Use Firma_Transportowa;

Create table if not exists Adresy
(
	idAdresu int not null auto_increment primary key,
    ulica varchar(50) not null,
    nrDomu varchar(7) not null,
    nrMieszkania int,
    kodPocztowy varchar(9),
    miejscowosc varchar(25)
);

Create table if not exists Osoby
(
	idOsoby int not null auto_increment primary key,
    idAdresu int references Adresy(idAdresu),
    imie varchar(20) not null,
    nazwisko varchar(30) not null,
    numerTelefonu varchar(12) not null,
    rola varchar(10)
);


Create table if not exists Pracownicy
(
	idPracownika int not null auto_increment primary key,
    idOsoby int references Osoby(idOsoby),
    login varchar(20) not null,
    haslo varchar(30) not null check(length(haslo)>8),
    wynagrodzenie float(6),
    dataZatrudnienia date not null,
    dataZwolenienia datetime
);

Create table if not exists Pojazdy
(
	idPojazdu int not null auto_increment primary key,
    numerRejestracyjny varchar(10) not null,
    rodzajPojazdu varchar(10) not null,
    markaPojazdu varchar(15) not null,
    modelPojazdu varchar(10) not null,
    sprawnoscPojazdu tinyint(1) not null default '1',
    dostepnoscPojazdu tinyint(1) not null default '1'
);

Create table if not exists Zlecenia
(
	idZlecenia int not null auto_increment primary key,
    idPracownika int references Pracownicy(idPracownika),
    idPojazdu int references Pojazdy(idPojazdu),
    idOsoby int references Osoby(idOsoby),
```

```
    miejsceOdbioru int references Adresy(idAdresu),
    miejsceDoreczenia int references Adresy(idAdresu),
    cena float(6) not null,
    dataZamowienia date not null,
    terminZlecenia date not null,
    czyOplacone tinyint(1) not null default '0',
    wagaPaczki float(6) not null,
    stanZamowienia varchar(20) not null default 'Nieoplacone',
    sciezkaDoPlikuFaktury varchar(40) not null

);

Create table if not exists Cennik
(
       cenaZaKm float(6) not null,
    cenaZaKg float(6) not null
);
```

5.2. Implementacja mechanizmów przetwarzania danych.

5.2.1. Widoki

Tworzymy dwa widoki - jeden pozwala na wyświetlenie nazwisk pracowników wraz z ich wynagrodzeniem, drugi natomiast umożliwia wyświetlenie listy zamówień.

```
create or replace view Lista_Pracownikow
as select osoby.imie, osoby.nazwisko, pracownicy.wynagrodzenie
from osoby join pracownicy on osoby.idOsoby=pracownicy.idOsoby;

create or replace view Lista_Zamowien
as select osoby.imie, osoby.nazwisko,concat(a1.ulica,'
',a1.nrDomu,'/',ifnull(a1.nrMieszkania,' '),', ',a1.miejscowosc) as
miejsceOdbioru,
concat(a2.ulica,' ',a2.nrDomu,'/',ifnull(a2.nrMieszkania,' '),', ',a2.miejscowosc)
as miejsceDoreczenia,zlecenia.terminZlecenia
from osoby
       join zlecenia on zlecenia.idOsoby = osoby.idOsoby
       join adresy a1 on a1.idAdresu = zlecenia.miejsceOdbioru
       join adresy a2 on a2.idAdresu = zlecenia.miejsceDoreczenia;
```

5.2.2. Triggery

Zaimplementowany został jeden trigger służący do automatycznej zmiany stanu zamówienia po zapłaceniu za nie.

```
delimiter $$
Create trigger Trigger_wRealizacji before update on zlecenia
for each row
begin
       if new.czyOplacone = '1' then
       set new.stanZamowienia = 'W realizacji';
    end if;
end;
$$
delimiter ;
```

5.2.3. Tworzenie indeksów.

Stworzone zostały podstawowe indeksy do usprawnienia przeszukiwania bazy danych. Indeks dataZamowienia został opisany jako malejący gdyż w pierwszej kolejności powinien zwracać aktualną datę.

```
create index osoba
on osoby(nazwisko, imie);

create index dataZamowienia
on zlecenia(dataZamowienia) desc;

create unique index numerTelefonu
on osoby(numerTelefonu);

create unique index numerRejestracyjny
on pojazdy(numerRejestracyjny);

create unique index login
on pracownicy(login);
```

5.3 Implementacja uprawnień i innych zabezpieczeń.

5.3.1. Tworzenie roli

Sworzyliśmy trzy role na podstawie tabeli z punktu 4.1.4

```
create role 'klient','pracownik','zarzadca';

grant insert,select,update,delete on zlecenia
to klient;

grant insert,update, select on adresy
to klient;

grant select on cennik
to klient;

grant insert,update, select on osoby
to klient;

grant insert,select,update,delete on pojazdy
to pracownik;

grant insert,select,update,delete on zlecenia
to pracownik;

grant select on adresy
to pracownik;

grant select,update on cennik
to pracownik;

grant select,update,delete on lista_zamowien
to pracownik;

grant all on pracownicy
to zarzadca;
```

```
grant all on pojazdy
to zarzadca;

grant all on zlecenia
to zarzadca;

grant all on adresy
to zarzadca;

grant all on cennik
to zarzadca;

grant all on lista_zamowien
to zarzadca;

grant all on lista_pracownikow
to zarzadca;
```

5.3.2 Tworzenie użytkowników.

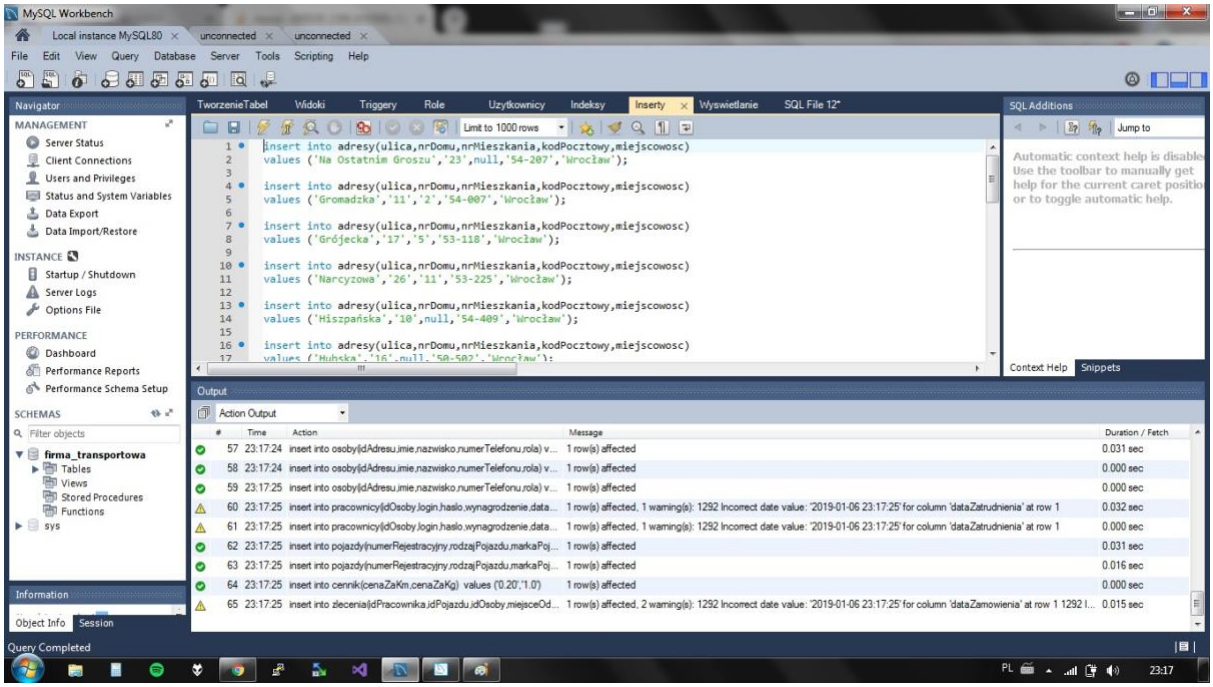
Stworzyliśmy po jednym użytkowniku dla każdej stworzonej wcześniej roli.

```
create user if not exists 'klient'@'localhost' default role klient;

create user if not exists 'pracownik'@'localhost' identified by 'pracownik'
default role pracownik;

create user if not exists 'zarzadca'@'localhost' identified by 'zarzadca' default
role zarzadca;
```

5.4. Testowanie bazy danych na przykładowych danych.

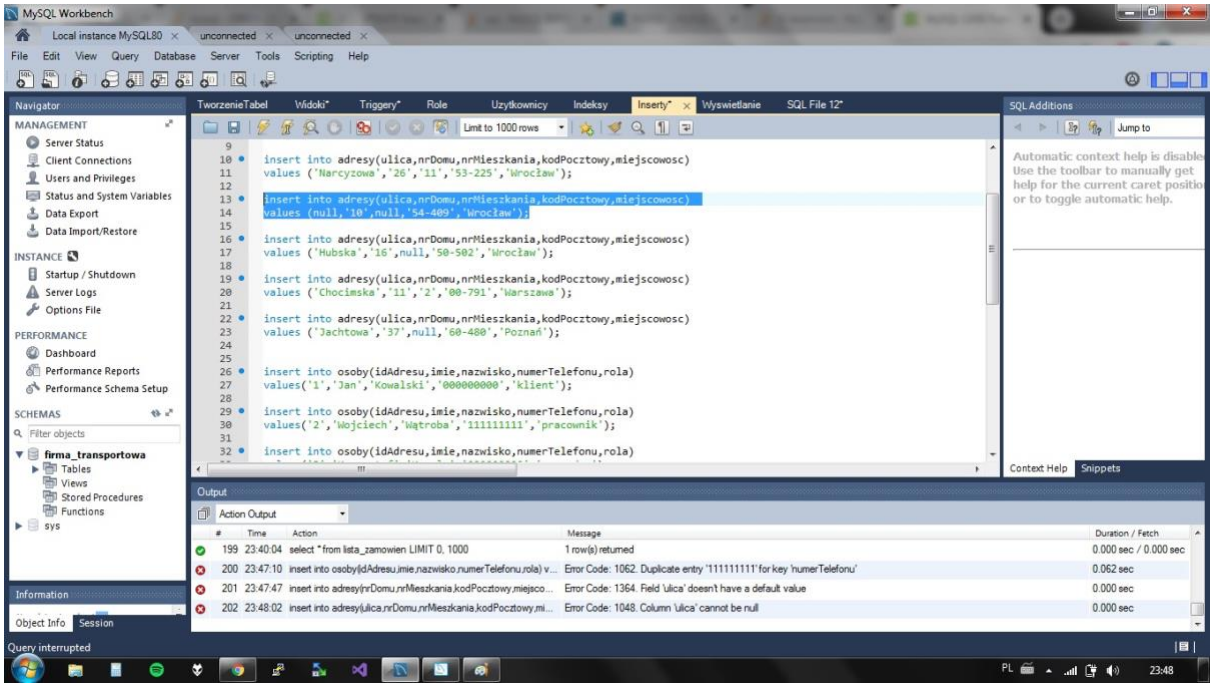
5.4.1. Wypełnianie tabel.

Tabele zostały wypełnione minimalną liczbą rekordów potrzebnych do przetestowania prawidłowego działania aplikacji.

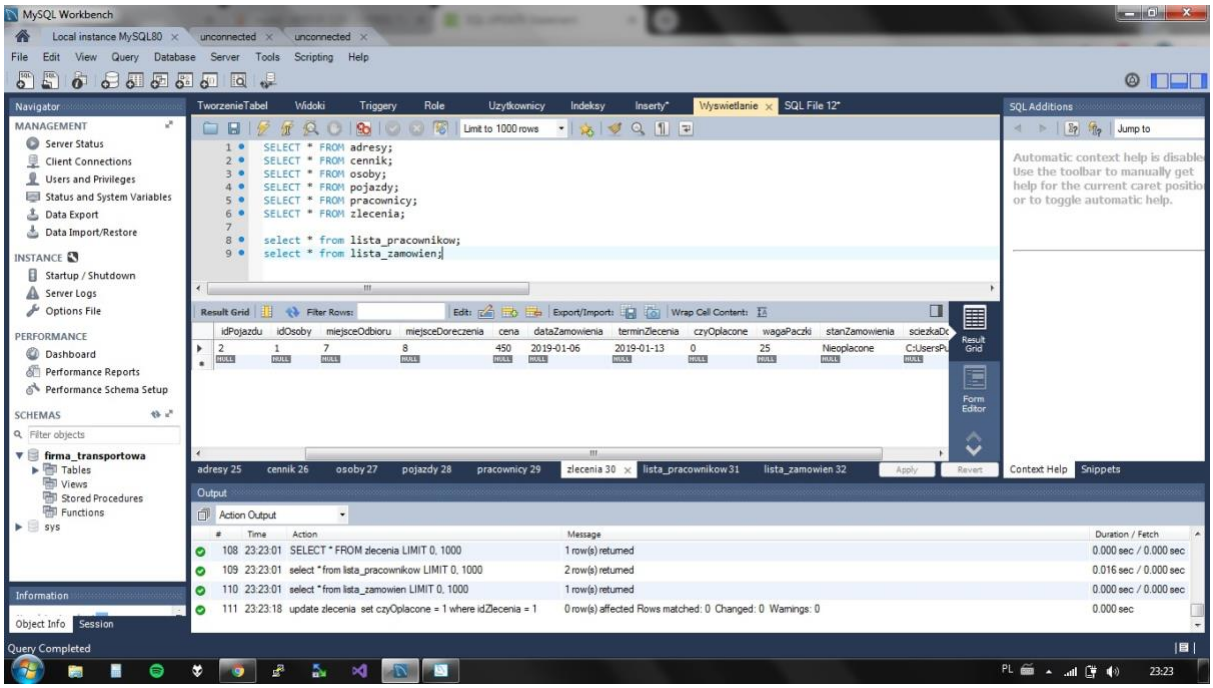


5.4.2. Test na wprowadzanie nulla.

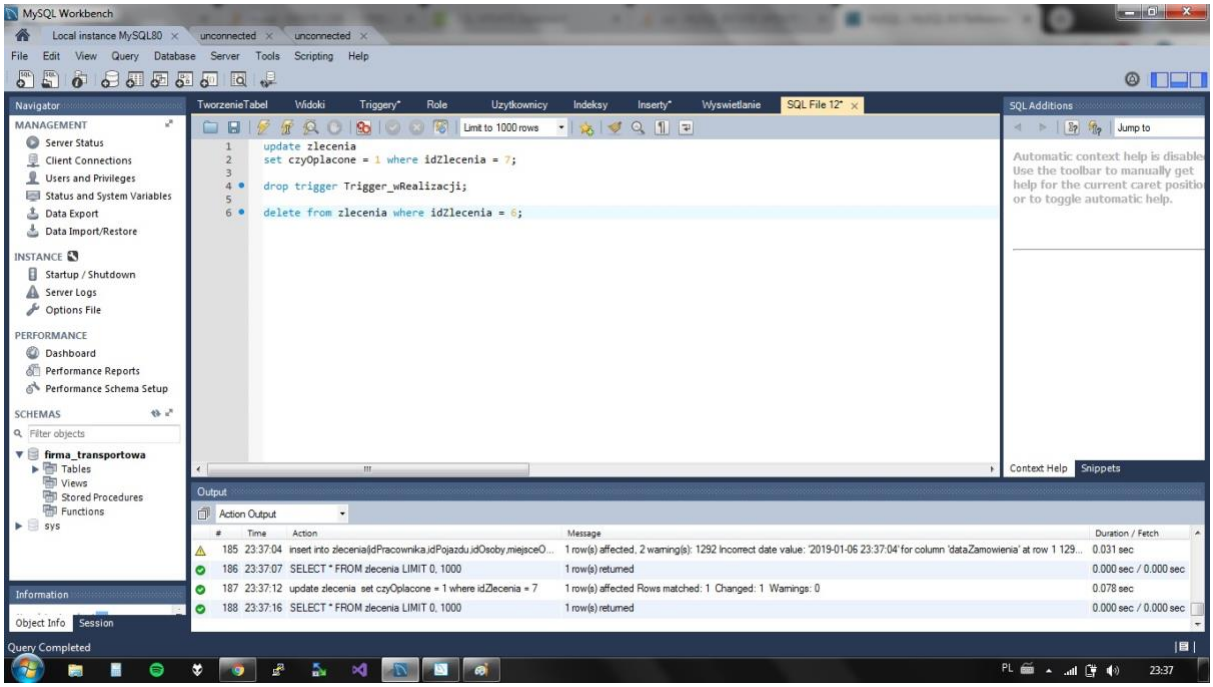
Test sprawdza działanie ograniczenia not null.



5.4.3. Testowanie triggera
5.4.3.1. Stan przed triggerem.
Przed zapłaceniem zamówienia jego stan jest ustawiony na „Nieopłacone”

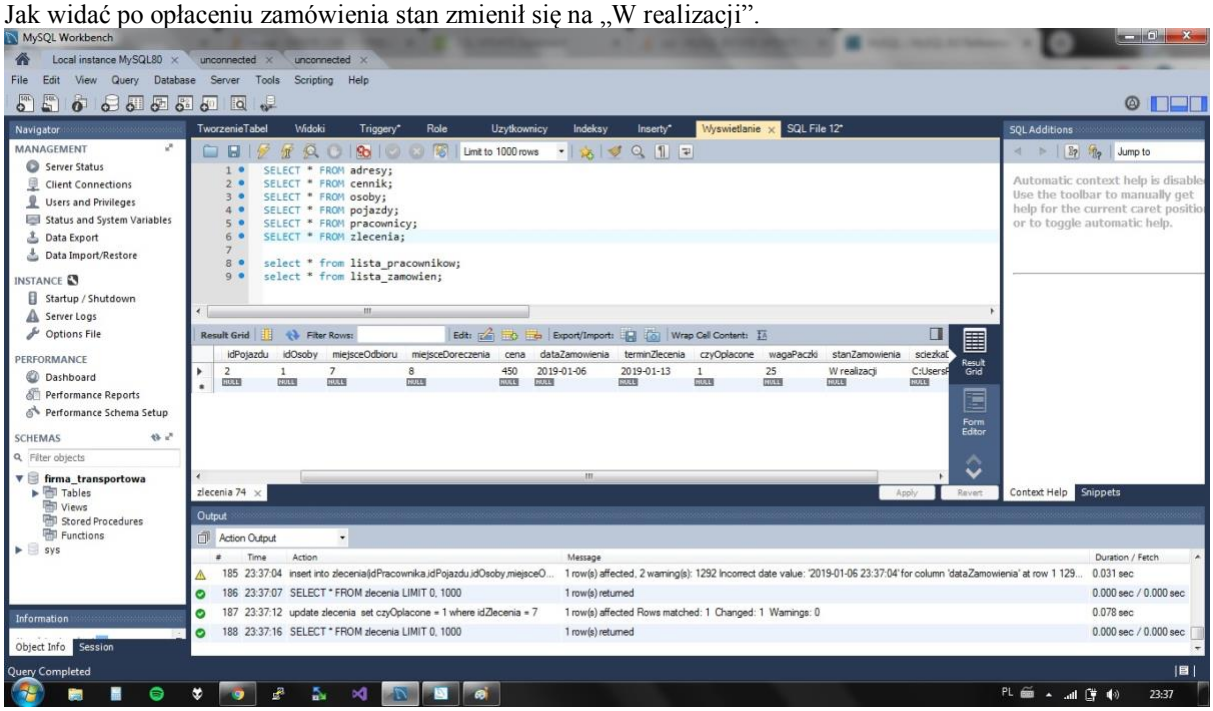


5.4.3.2. Inicjalizacja triggera.
Trigger działa w momencie opłacenia zamówienia.

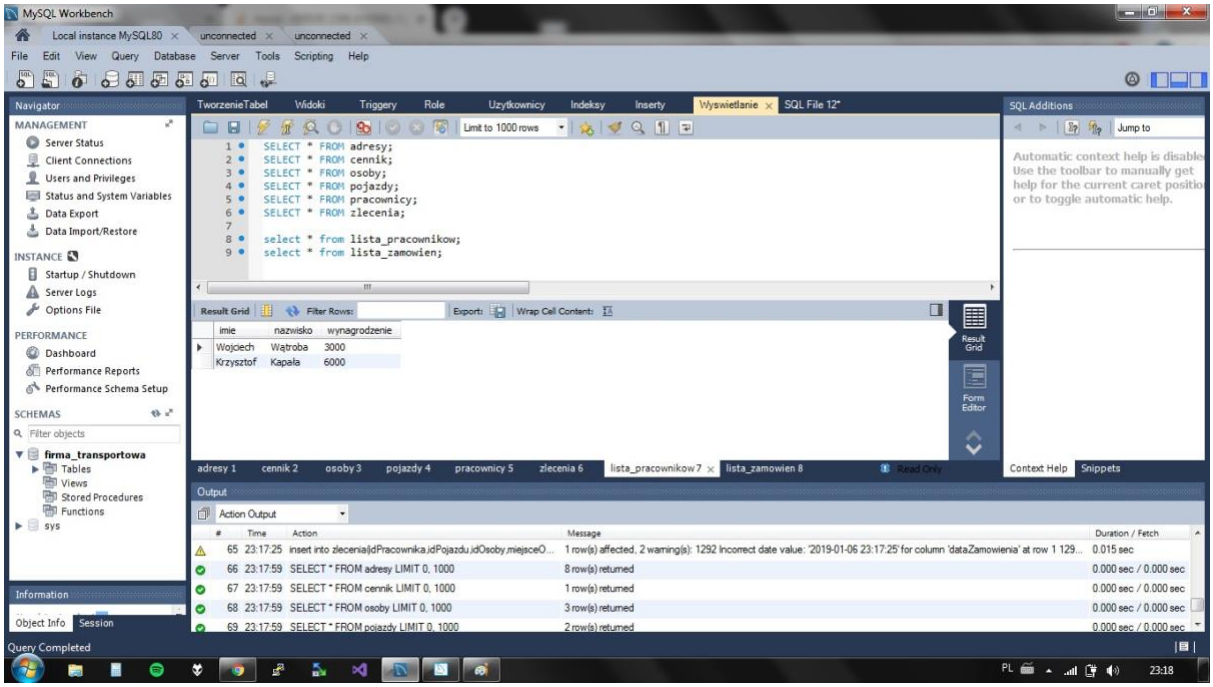


5.4.3.3. Stan po triggerze.

Jak widać po opłaceniu zamówienia stan zmienił się na „W realizacji”.

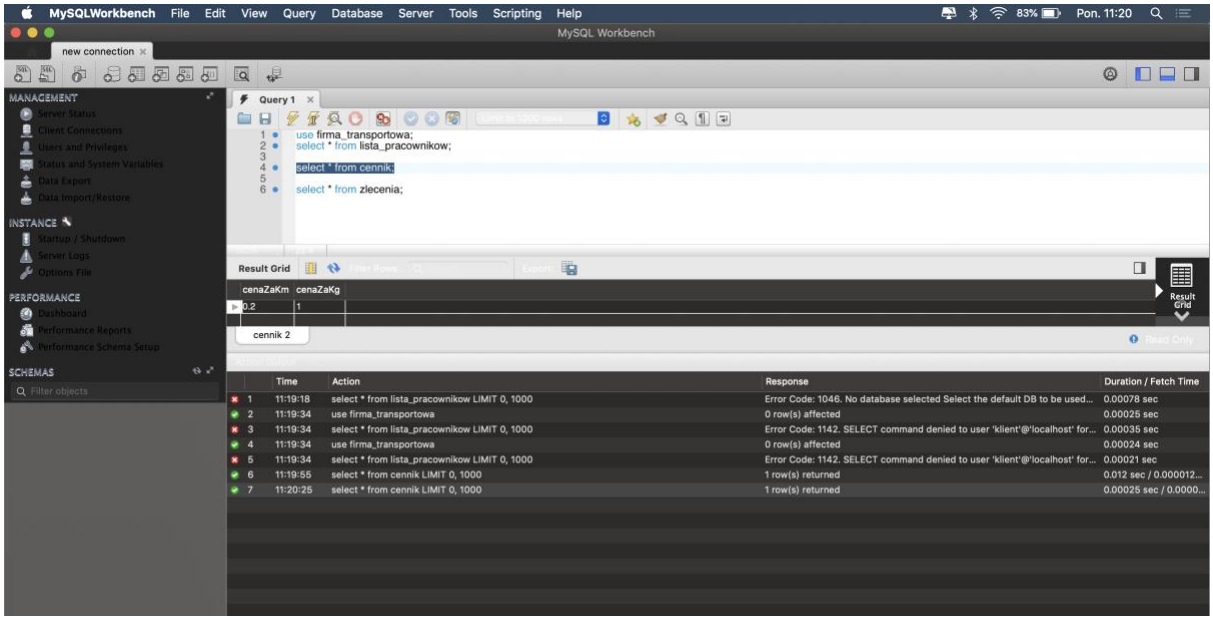


5.4.4. Wyświetlanie widoku.



5.4.5. Testowanie uprawnień

Sprawdzenie czy klient nie ma możliwości przeszukiwania listy pracowników.



6. Implementacja i testy aplikacji

6.1. Instalacja i konfigurowanie systemu.

Aby zainstalować system, należy uruchomić nowy serwer MySql, następnie za pomocą edytora sql np. MySql Workbench, uruchomić załączone skrypty sql w kolejności: TworzenieTabel.sql, Widoki.sql, Triggery.sql, Role.sql, Użytkownicy.sql, Indeksy.sql, Inserty.sql. Następnie uruchomić plik FirmaTransportowa.exe

6.2. Instrukcja użytkowania aplikacji.

Po uruchomieniu aplikacji otworzy się okno logowania. Należy wybrać opcje klient lub pracownik a następnie wpisać poprawny login i hasło, otrzymane od administratora systemu.

Użytkownik może złożyć nowe zamówienie lub wyświetlić wcześniej złożone. Po wybraniu opcji „Zapłać" zostanie przekierowany na stronę banku w celu dokonania płatności.

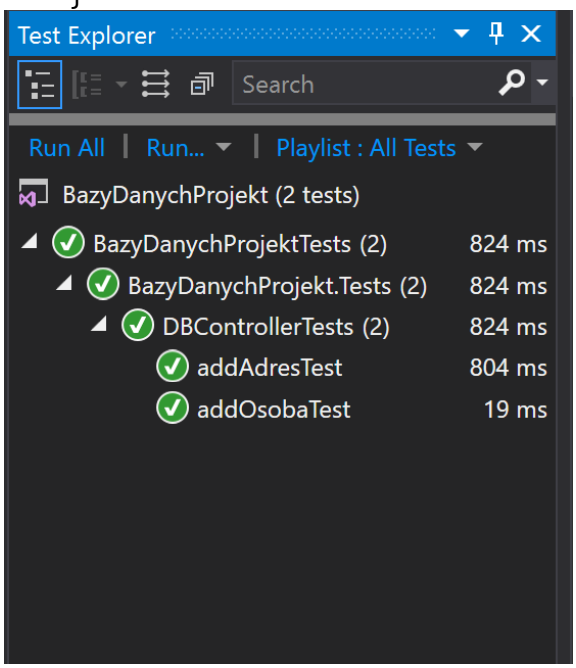
Pracownik może przeglądać listę wszystkich złożonych zamówień, przypisać do nich pojazd a po tym, wysłać zamówienie do realizacji.

6.3. Testowanie opracowanych funkcji systemu.

Testowanie dodawania adresu i dodawania osoby. Uruchomione zostały testy jednostkowe, które przeszły poprawnie.

```
[TestClass()]
    public class DBControllerTests
    {
        [TestMethod()]
        public void addAdresTest()
        {
            DBController.loginToDb("klient", "");

            string ulica = "Jakas Ulica";
            string nrDomu = "12";
            string nrMieszkania = "4";
            string kodPocztowy = "45-231";
            string miejscowosc = "Wroclaw";

            Adres adres = new Adres(ulica, nrDomu,
                Int32.Parse(nrMieszkania),
                kodPocztowy,
                miejscowosc);

            Assert.AreEqual(true, DBController.addAdres(adres));
        }

        [TestMethod()]
        public void addOsobaTest()
        {
            DBController.loginToDb("klient", "");

            Osoba osoba = new Osoba();
            osoba.idAdresu = 2;
            osoba.imie = "Jan";
            osoba.nazwisko = "Nowak";
            osoba.numerTelefonu = "34203969";
```

```
            Assert.AreEqual(true, DBController.addOsoba(osoba));
        }
    }
```



6.4 Omówienie wybranych rozwiązań programistycznych.

6.4.1. Implementacja interfejsu dostępu do bazy danych.

Logowanie do bazy danych z użyciem klasy MySqlConnection.

```
public static bool loginToDb(String login, String password)
{
    String conStr = "SERVER=localhost;DATABASE=firma_transportowa;uid =" + login + ";Password=" + password + ";";
    Console.WriteLine(conStr);
    MySqlConnection sqlConnection = new MySqlConnection(conStr);
    try
    {
        sqlConnection.Open();
        connectionString = conStr;
        user = login;

        sqlConnection.Close();
        return true;
    }
    catch (Exception ex)
    {
        MessageBox.Show("Zły login lub hasło!");
    }

    return false;
}
```

Przykładowe pobranie danych z bazy z użyciem klas: MySqlConnection, MySqlDataReader oraz zapisanego wcześniej stringa z danymi logowania.

```
public static List<Zamowienie> getListaZamowien()
{
    MySqlConnection sqlConnection = new MySqlConnection(connectionString);
    MySqlDataReader reader = null;
    try
    {
        sqlConnection.Open();
        String sql = "select * from zlecenia";
        MySqlCommand cmd = new MySqlCommand(sql, sqlConnection);
        reader = cmd.ExecuteReader();
        Dictionary<string, object> dict = new Dictionary<string, object>();
        List<Zamowienie> zamowienia = new List<Zamowienie>();
        while (reader.Read())
        {
            for (int lp = 0; lp < reader.FieldCount; lp++)
            {
                dict.Add(reader.GetName(lp), reader.GetValue(lp));
            }
            Zamowienie tmp = new Zamowienie(dict);
            zamowienia.Add(tmp);
            dict = new Dictionary<string, object>();
        }

        sqlConnection.Close();
        return zamowienia;
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception: " + ex);
    }

    return null;
}
```

Przykładowe dodanie danych do bazy z użyciem klas MySqlConnection oraz MySqlCommand.

```
public static bool addAdres(Adres adres)
{
    MySqlConnection sqlConnection = new MySqlConnection(connectionString);
    try
    {
        sqlConnection.Open();
        String sql = "insert into adresy(ulica,nrDomu,nrMieszkania,kodPocztowy,miejscowosc)
        values(@ulica, @nrDomu, @nrMieszkania, @kodPocztowy, @miejscowosc)";
        MySqlCommand cmd = new MySqlCommand(sql, sqlConnection);
        cmd.Prepare();
        cmd.Parameters.AddWithValue("@ulica", adres.ulica);
        cmd.Parameters.AddWithValue("@nrDomu", adres.nrDomu);
        cmd.Parameters.AddWithValue("@nrMieszkania", adres.nrMieszkania);
        cmd.Parameters.AddWithValue("@kodPocztowy", adres.kodPocztowy);
        cmd.Parameters.AddWithValue("@miejscowosc", adres.miejscowosc);
        cmd.ExecuteNonQuery();

    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception: " + ex);
        return false;
    }
    finally
    {
        if (sqlConnection != null)
        {
            sqlConnection.Close();
        }
    }
    return true;
}
```

6.4.2. Implementacja wybranych funkcjonalności systemu.

Logowanie do systemu i wybranie odpowiedniej roli.

```
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            String login = loginTextField.Text;
            String password = passwordTextField.Text;

            if (DBController.loginToDb(login, password))
            {
                if (klientCheckBox.IsChecked.Value)
                {
                    KlientMenu klient = new KlientMenu();
                    klient.Show();
                }
                else if (pracownikCheckBox.IsChecked.Value)
                {
                    PracownikMenu pracownik = new PracownikMenu();
                    pracownik.Show();
                } else
                {
                    MessageBox.Show("Nie wybrano roli!");
                    return;
                }
                Close();
            }
        }
```

Zapisanie nowego adresu w zależności od wybranego trybu adresu (osoby, doręczeczenia, odbioru).

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    try
    {
        Adres nowyAdres = new Adres(ulicaTextBox.Text, nrDomuTextBox.Text,
            Int32.Parse(nrMieszkaniaTextBox.Text),
            kodPocztowyTextBox.Text,
            miejscowoscTextBox.Text);
        DBController.addAdres(nowyAdres);
        if (mode == 0)
        {
            cont.osoba.idAdresu = DBController.getAdresId(nowyAdres);
        }
        else if (mode == 1)
        {
            cont.zamowienie.idAdresuOdbioru = DBController.getAdresId(nowyAdres);
        }
        else
        {
            cont.zamowienie.idAdresuDoreczenia = DBController.getAdresId(nowyAdres);
        }

        Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Zły format danych!");
    }
}
```

Wyświetlenie zamówienia klientowi, jeżeli wcześniej je złożył.

```
public ZamowienieController()
{
    InitializeComponent();

    zamowienie = DBController.getZamowienieForUser();
    if (zamowienie != null)
    {
        osoba = DBController.getOsoba(zamowienie.idOsoby);
        imieTextBox.Text = osoba.imie;
        nazwiskoTextBox.Text = osoba.nazwisko;
        nrTelefonuTextBox.Text = osoba.numerTelefonu;
        wagaPaczkiTextBox.Text = zamowienie.wagaPaczki.ToString();
    }
    else
    {
        zamowienie = new Zamowienie();
        osoba = new Osoba();
    }
}
```

6.4.3. Implementacja mechanizmów bezpieczeństwa

Logowanie do bazy danych wraz z zabezpieczeniem przed wprowadzeniem błędnych danych logowania.

```
public static bool loginToDb(String login, String password)
{
    String conStr = "SERVER=localhost;DATABASE=firma_transportowa;uid =" + login + ";Password=" + password + ";";
    Console.WriteLine(conStr);
    MySqlConnection sqlConnection = new MySqlConnection(conStr);
    try
    {
        sqlConnection.Open();
        connectionString = conStr;
        user = login;

        sqlConnection.Close();
        return true;
    }
    catch (Exception ex)
    {
        MessageBox.Show("Zły login lub hasło!");
    }

    return false;
}
```

7. Podsumowanie i wnioski

Wykonane zadanie projektowe umożliwiło nam odświeżenie sobie wiedzy z poprzednich semestrów oraz nauczenia się kompletnie nowych zagadnień dotyczących projektowania oraz implementowania baz danych jak i systemów je obsługujących.

Stworzony przez nas system system zarządzania firmą transportową spełnia wszystkie założenia projektowe oraz wymagania do funkcjonowania w rzeczywistym świecie.

Zarówno system bazodanowy, jak i aplikacja dostępowa stanowią duże uproszczenie świata rzeczywistego.